PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-179964
(43)Date of publication of application : 12.07.1996

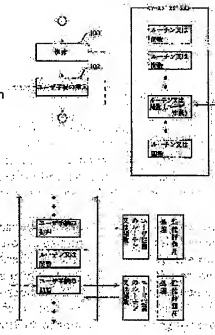
(51)Int.Cl. G06F 11/28

(21)Application number : 06-324325
(22)Date of filing : 27.12.1994

(71)Applicant : FUJITSU LTD
(72)Inventor : OHASHI TAKAMI
AOKI MASAKI

(54) PROGRAM CHECKING METHOD

(57)Abstract:
PURPOSE: To remarkably reduce quantity to be described additionally in a source program by inserting the processing of user procedure by which a routine or function of user definition is called into another routine or the just before position or immediately behind position of the function.
CONSTITUTION: A user who desires to obtain information with respect to each routine or function inserts and defines only a specific routine or function (the routine or function of user definition) into the source program (100). When the routine or function of user definition is described, the processing of user procedure is inserted into another routine or the just before position or immediately behind position of the function (102). In such a way, the routine or function of user definition is called by the processing of user procedure, therefore, the user can confirm the information with respect to a sub-routine or function without describing processing for program check at every sub-routine or function.



LEGAL STATUS

[Date of request for examination]	28.08.2000
[Date of sending the examiner's decision of rejection]	23.07.2002
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	
[Date of registration]	
[Number of appeal against examiner's decision of rejection]	2002-015896
[Date of requesting appeal against examiner's	21.08.2002

decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平8－179964

(43)公開日　平成８年(1996)７月12日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　11/28	３１５	7313－5Ｂ		

審査請求　未請求　請求項の数２　ＯＬ　（全 10 頁）

(21)出願番号　特願平6－324325

(22)出願日　平成６年(1994)12月27日

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号

(72)発明者　大橋　勝美
静岡県静岡市伝馬町16番地の３　株式会社富士通静岡エンジニアリング内

(72)発明者　青木　正樹
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内

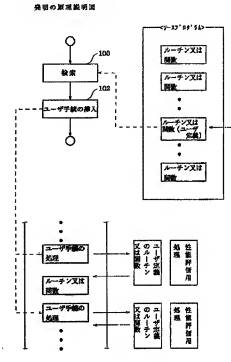
(74)代理人　弁理士　伊藤　儀一郎

(54)【発明の名称】　プログラムチェック方法

(57)【要約】

【目的】　本発明は、プログラムをそのデバッグやチューンのためにチェックする方法に関し、ソースプログラムへ追加記述する量を大幅に削減することが可能となる方法の提供を目的とする。

【構成】　ソースプログラムに含まれた複数のルーチン又は関数から他のルーチン又は関数に関する情報の収集処理が内容とされたユーザ定義のルーチン又は関数を検索し（１００）、ユーザ定義のルーチン又は関数が呼び出されるユーザ手続きの処理を他のルーチン又は関数の直前位置と直後位置へ挿入する（１０２）。



1

【特許請求の範囲】

【請求項１】　ソースプログラムに含まれた複数のルーチン又は関数から他のルーチン又は関数への移行／復帰に関する情報の収集処理を内容とするユーザ手続きのルーチン又は関数を検索し、

上記ユーザ手続きのルーチン又は関数の呼び出される処理を他のルーチン又は関数の直前位置と直後位置へ挿入する、

ことを特徴としたプログラムチェック方法。

【請求項２】　前記情報収集処理は性能評価処理であることを特徴とする請求項１項記載のプログラムチェック方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、プログラムをそのデバッグやチューンのためにチェックする方法に関するものである。

【０００２】プログラムのデバッグやチューンが行なわれる場合、プログラム内に存在の各ルーチン，関数に関する情報を出力させる内容がソースプログラムにあらかじめ記述される。

【０００３】

【従来の技術】図２において最初の任意処理（ステップ２０２）が終了すると、前処理（ステップ２０４）が行なわれてから、サブルーチン（関数）が呼び出される（ステップ２０６）。

【０００４】呼び出されたサブルーチンでは前処理（ステップ２０８）が行われてから、その本体処理（ステップ２１０）が開始され、サブルーチンの本体処理（ステップ２１０）が終了すると、サブルーチンの後処理（ステップ２１２）が行われてから任意処理（ステップ２１４）が開始される。

【０００５】サブルーチンの前処理（ステップ２０４）に関するソースプログラム部分ではサブルーチン呼び出しの行番号を採取し、サブルーチン呼び出し直前までの経過時間を取得し、サブルーチン内における前処理（ステップ２０８）の初期化を行なう内容が記述される。

【０００６】また、サブルーチン内の前処理（ステップ２０８）に関するソースプログラム部分ではサブルーチン名や引数を出力する内容が記述され（図３，図４参照）、サブルーチンの後処理（ステップ２１２）に関するソースプログラム部分ではサブルーチン呼び出し後までの経過時間やサブルーチン内の演算数を取得して出力する内容が記述される。

【０００７】このプログラムを作成したユーザは出力された情報を用いて同プログラムをデバッグし、チューンする。

【０００８】

【発明が解決しようとする課題】従来においては、プログラムのデバッグやチューンに際してサブルーチン（関数）毎にプログラムチェック用の処理（前処理，サブルーチン内前処理，後処理）をソースプログラムにあらかじめ記述し、また、その後に不要となったこれらの記述を削除することが必要となるので、プログラムの開発に時間，労力が費やされていた。

2

【０００９】本発明は上記従来の事情に鑑みてなされたものであり、その目的は、ソースプログラムへ追加記述する量を大幅に削減することが可能となる方法を提供することにある。

【００１０】

【課題を解決するための手段】図１において第１発明では、ソースプログラムに含まれた複数のルーチン又は関数から他のルーチン又は関数に関する情報の収集処理が内容とされたユーザ定義のルーチン又は関数を検索し（ステップ１００）、ユーザ定義のルーチン又は関数が呼び出されるユーザ手続きの処理を他のルーチン又は関数の直前位置と直後位置へ挿入する（ステップ１０２）。

【００１１】図１において第２発明では、ソースプログラムに含まれた複数のルーチン又は関数から他のルーチン又は関数に関する情報の収集処理が内容とされたユーザ定義のルーチン又は関数を検索し（ステップ１００）、ユーザ定義のルーチン又は関数と他のルーチン又は関数の性能評価用処理とのうちいずれか又は双方がユーザ指示に従って呼び出されるユーザ手続きの処理を他のルーチン又は関数の直前位置と直後位置へ挿入する（ステップ１０２）。

【００１２】

【作用】各ルーチン又は各関数に関する情報を得ようとするユーザはそのソースプログラム内に特定のルーチン又は関数（ユーザ定義のルーチン又は関数）のみを挿入定義する。

【００１３】このユーザ定義のルーチン又は関数が記述されると、ユーザ手続きの処理が他のルーチン又は関数の直前位置と直後位置へ挿入される。第１発明においては、ユーザ手続きの処理でユーザ定義のルーチン又は関数が呼び出され、したがってユーザは、サブルーチン又は関数毎にプログラムチェック用の処理を記述することなく、それらサブルーチン又は関数に関する情報を確認することが可能となる。

【００１４】第２発明においては、ユーザ定義のルーチン又は関数と他のルーチン又は関数の性能評価用処理とのうちいずれか又は双方がユーザ指示に従いユーザ手続きの処理で呼び出される。

【００１５】性能評価用処理はルーチン又は関数を呼び出してから経過した時間やルーチン又は関数内の演算数を取得して出力するなどの内容とされ、ユーザは自己が定義したルーチン又は関数と性能評価用処理とを任意に選択できる。

【００１６】なお、ユーザ手続き処理，性能評価用処理

はソース，オブジェクトなどの形式であらかじめ用意され、また、ファイル処理，コンパイル処理などでソース，オブジェクトのプログラムファイルへ挿入できる。

【００１７】プログラムのデバッグやチューンの作業が完了した場合は、そのソースプログラムからユーザ定義のルーチン又は関数を削除するなどにより、プログラムを高速化できる。

【００１８】

【実施例】図５において、キーボード５００，ディスプレイ５０２，ハードディスク５０４がコンピュータ本体５０６に接続されており、ユーザはキーボード５００を操作し、ディスプレイ５０２の表示を確認する。

【００１９】ハードディスク５０４にはエディタ，コンパイラ，ダイナミックリンクルーチンのライブラリが用意されており、ユーザはエディタを起動してソースプログラムを作成し、コンパイラを用いてソースプログラムからそのロードモジュールを生成し、ロードモジュールはダイナミックリンクルーチンを参照する。

【００２０】図６，図７にはソースプログラムの例が示されており、ユーザはサブルーチンＵＳＥＲ＿ＤＥＦＩＮＥＤ＿ＰＲＯＣ，関数ｕｓｅｒ＿ｄｅｆｉｎｅｄ＿ｐｒｏｃ＿（ユーザ定義のルーチン又は，関数）のみをソースプログラムへあらかじめ追加記述する。

【００２１】それら図６，図７において、文字列Ｎ，ｎはサブルーチンＳＵＢ，関数ｓｕｂを示し、変数Ｌ，ｌは行番号を示し、整数フラグＦ，ｆはサブルーチンＵＳＥＲ＿ＤＥＦＩＮＥＤ＿ＰＲＯＣ，関数ｕｓｅｒ＿ｄｅｆｉｎｅｄ＿ｐｒｏｃ＿の動作モード（表示出力の内容）を決定する。

【００２２】図８では本実施例におけるコンパイル処理がフローチャートを用いて説明されており、同図において、最初にデバッグのコンパイルスイッチがオンされたか否かが判断される（ステップ８００）。

【００２３】デバッグのコンパイルスイッチがオンされたことを示す判断結果が得られた場合には、サブルーチン（又は関数）の入口位置と出口位置にライブラリ呼出命令が挿入される（ステップ８０１：ユーザ手続きの処理が他のルーチン又は関数の直前位置と直後位置へ挿入される）。

【００２４】デバッグのコンパイルスイッチがオンされなかったことを示す判断結果が得られた場合には、サブルーチン（又は関数）の入口位置と出口位置にライブラリ呼出命令が挿入されずることなく、ユーザ定義のルーチン又は関数（図６，図７においては、サブルーチンＵＳＥＲ＿ＤＥＦＩＮＥＤ＿ＰＲＯＣ，関数ｕｓｅｒ＿ｄｅｆｉｎｅｄ＿ｐｒｏｃ＿）がソースプログラムへ追加記述されているか否かにかかわらず、単に通常のロードモジュール生成処理が行われる（ステップ８０３）。

【００２５】デバッグのコンパイルスイッチがオンされたことを示す判断結果が得られてライブラリ呼出命令の挿入が行なわれると、スタティックリンクをすべきか否かが判断される（ステップ８０４）。

【００２６】スタティックリンクをすべき旨の判断結果が得られた場合にはコンパイラ側であらかじめ用意されたデバッグ関数（性能評価用処理：ルーチン又は関数を呼び出してから経過した時間やルーチン又は関数内の演算数を取得して出力するなどの内容）を結合する処理が行なわれる（ステップ８０５）。

【００２７】さらに、ユーザ定義のルーチン又は関数がソースまたはオブジェクトの形式で存在するか否かが判断され（ステップ８０６），存在することを示す判断結果が得られた場合にはこれを含むロードモジュールが生成される（ステップ８０７，８０３）。

【００２８】また存在しないことを示す判断結果が得られた場合には、何らの処理も行なわれない内容であってユーザ定義のルーチン又は関数に付された名称のルーチン又は関数を含むロードモジュールが生成される（ステップ８０８，８０３）。

【００２９】そしてダイナミックリンクをすべき旨の判断結果が得られた場合には（ステップ８０４）、ユーザ定義のルーチン又は関数がソースプログラムの本体と分けられてコンパイルされ（ステップ８０３），ダイナミックリンクのルーチンとされる。

【００３０】図９ではロードモジュールの実行処理がフローチャートを用いて説明されており、同図において、最初にダイナミックリンクが行なわれるか否かが判断される（ステップ９００）。

【００３１】ダイナミックリンクが行なわれない場合にはプログラムの起動時にライブラリデバッグのスイッチがオンされたか否かが判断され（ステップ９０１），オンされたときにはデバッグ関数が、オンされなかったときにはユーザ定義のルーチン又は関数が選択される（ステップ９０２，９０３）。

【００３２】ダイナミックリンクが行なわれる場合にもプログラムの起動時にライブラリデバッグのスイッチがオンされたか否かが判断され（ステップ９０４），また、ユーザ定義のルーチン又は関数から生成されたダイナミックリンクルーチンの存在有無が判断される（ステップ９０５）。

【００３３】ダイナミックリンクが行なわれる場合で、プログラムの起動時にライブラリデバッグのスイッチがオンされたときには、ダイナミックリンクルーチンのデバッグ関数が参照される（ステップ９０６）。

【００３４】ダイナミックリンクが行なわれる場合で、プログラムの起動時にライブラリデバッグのスイッチがオンされ、かつ、ユーザ定義のルーチン又は関数から生成されたダイナミックリンクルーチンが存在するときには、このユーザ定義のルーチンが参照される（ステップ９０７）。

【００３５】ダイナミックリンクが行なわれる場合で、

プログラムの起動時にライブラリデバッグのスイッチがオンされ、かつ、ユーザ定義のルーチン又は関数から生成されたダイナミックリンクルーチンが存在しないときには、何らの処理も行なわれない内容であってユーザ定義のルーチン又は関数に付された名称のダイナミックリンクルーチンが参照される（ステップ９０８）。

【００３６】図１０では本実施例の作用（図２参照）が説明されており、同図において、サブルーチン処理（ステップ２１０）の呼出が行なわれると（ステップ２０６）、そのサブルーチン処理（ステップ２１０）の前に、ライブラリの入口処理（ライブラリ呼出命令／ユーザ手続きの処理：ステップ１０００）で、ユーザ定義のサブルーチン（ステップ１００２）又はデバッグ処理（ステップ１００４）が起動される。

【００３７】サブルーチン処理（ステップ２１０）はユーザ定義のサブルーチン（ステップ１００２）又はデバッグ処理（ステップ１００４）が終了してから開始され、そのサブルーチン処理（ステップ２１０）が終了すると、ライブラリの出口処理（ライブラリ呼出命令／ユーザ手続きの処理：ステップ１００６）で、ユーザ定義のサブルーチン（ステップ１００２）又はデバッグ処理（ステップ１００４）が起動される。

【００３８】以上のように、プログラム内の各ルーチン又は各関数に関する情報を得るためにそのソースプログラムへ追加して記述すべきものが特定のルーチン又は関数（ユーザ定義のサブルーチン又は関数）の部分のみとなるので、デバック作業やチューニング作業の際におけるユーザの負担が大幅に軽減される（図２および図１０参照）。

【００３９】さらに、各ルーチン又は各関数に関する情報を表示出力させるか各ルーチン又は各関数の実行時間や演算数を表示出力させるかをプログラム起動時のオプションスイッチで選択できる。

【００４０】そして、コンパイルスイッチを変更するのみで、これら出力を抑止しながら、プログラムを高速化することが可能となる。また図６，図７のフラグ値を変更することで、各ルーチンは各関数に関する情報の表示*

＊内容を適切なものへ変更できる。

【００４１】

【発明の効果】以上説明したように本発明によれば、プログラム内の各ルーチン又は各関数に関する情報を表示出力させたり各ルーチン又は各関数の実行時間や演算数を表示出力させるためにソースプログラムへ追加記述すべきものが特定のルーチン又は関数（ユーザ定義のサブルーチン又は関数）についてのみで、わずかな量となることから、デバッグ作業やチューニング作業の際におけるユーザの負担を大幅に軽減してプログラムの開発効率を著しく高めることが可能となる。

【００４２】また、高速なプログラムを実行するためには、該プログラムを再翻訳するだけで良いため、その面からもユーザの負担を軽減することができる。

【図面の簡単な説明】

【図１】発明の原理説明図である。

【図２】従来技術の説明図である。

【図３】従来におけるソースプログラムの説明図である。

【図４】従来におけるソースプログラムの説明図である。

【図５】実施例の構成説明図である。

【図６】実施例におけるソースプログラムの説明図である。

【図７】実施例におけるソースプログラムの説明図である。

【図８】実施例のコンパイル処理を説明するフローチャートである。

【図９】実施例のプログラム実行処理を説明するフローチャートである。

【図１０】実施例の作用説明図である。

【符号の説明】

５００　キーボード

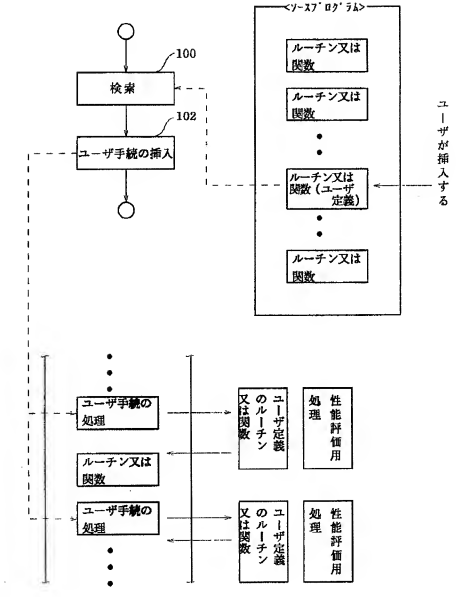
５０２　ディスプレイ

５０４　ハードディスク

５０６　コンピュータ本体

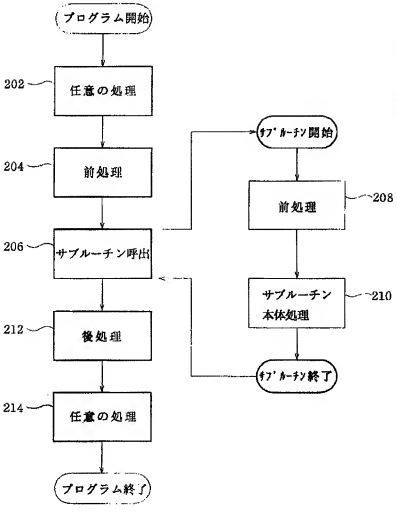
【図１】

発明の原理説明図


<ソースプログラム>
ルーチン又は関数
ルーチン又は関数
ルーチン又は関数（ユーザ定義）
ルーチン又は関数
ユーザが挿入する
100
検索
102
ユーザ手続の挿入
ユーザ手続の処理
ルーチン又は関数
ユーザ手続の処理
ユーザ定義のルーチン又は関数
性能評価用処理
ユーザ定義のルーチン又は関数
性能評価用処理

【図２】

従来技術の説明図



【図７】

実施例における１ステッププログラムの説明図

＜C言語で関数の情報を出力する例＞

```
main(){
        integer i;
        :
        sub(i);
        :
}
sub(i){
        int i;
        :
        :
}
user_defined_proc_(f,n,i){
        int *f,*i;
        char *n;
}
        int k;
        char name[5];
        if(*f=0)
                printf("Program start¥n");
        else if(*f=1)
                printf("End prosram¥n");
        else if(*f=2){
                printf("Proc start¥n");
                for(k=0;k<4;k++)
                        s[k]=*(n+k);
                s[k]='¥0';
                printf("name=%s¥n",s);
}
        else if(*f=3){
                printf("Proc end¥n");
                for(k=0;k<4;k++)
                        s[k]=*(n+k);
        s[k]='¥0';
        printf("name= %s¥n",s);
}
```

【図３】

従来技術におけるソースプログラムの説明図

＜ＦＯＲＴＲＡＮでサブルーチンの情報を出力する例＞

```
PROGRAM TEST
INTEGER I
.
.
I=100
WRITE(6,*)'Line No. = 6'
CALL SUB(I)
.
.
END

SUBROUTINE SUB(I)
INTEGER I

WRITE(6,*)'NAME=','SUB'
WRITE(6,*)'I=',I
.
.
END
```

【図４】

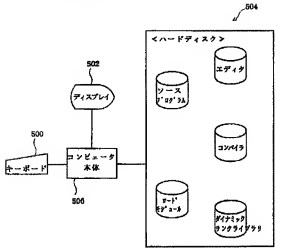
従来技術におけるソースプログラムの説明図

＜Ｃ言語で関数の情報を出力する例＞

```
main(){
        Integer I;
        .
        .
        I=100
        printf("Line no  = 6¥n");
        sub(I)
        .
        .
}
sub(I){
        Integer I;

        printf("name = %s¥n","SUB");
        printf("I = %d¥n",I);
        .
        .
}
```

【図５】

実施例の構成概略図



【図６】

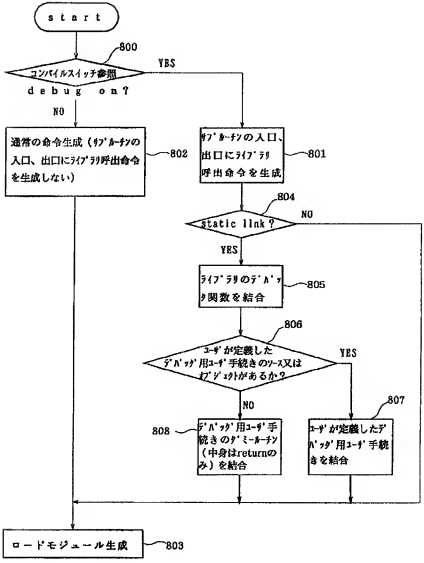
実施例におけるソースプログラムの説明図

＜ＦＯＲＴＲＡＮでサブルーチンの情報を出力する例＞

```
PROGRAM TEST
INTEGER I
:
CALL SUB(I)
:
END

SUBROUTINE SUB(I)
INTEGER I

:
:
END


SUBROUTINE USER_DEFINED_PROC(F,N,L)
INTEGER F,L
CHARACTER*(*)N
CHARACTER*8 NAME

NAME=N
IF(F.EQ.0) THEN
        WRITE(*,*)'PROGRAM START'
ELSEIF(F.EQ.1) THEN
        WRITE(*,*)'PROGRAM END'
ELSEIF(F.EQ.2) THEN
        WRITE(*,*)'PROC START:',NAME,'SSN:',L
ELSEIF(F.EQ.3) THEN
        WRITE(*,*)'PROC END:',NAME,'SSN:'L
ENDIF
END
```

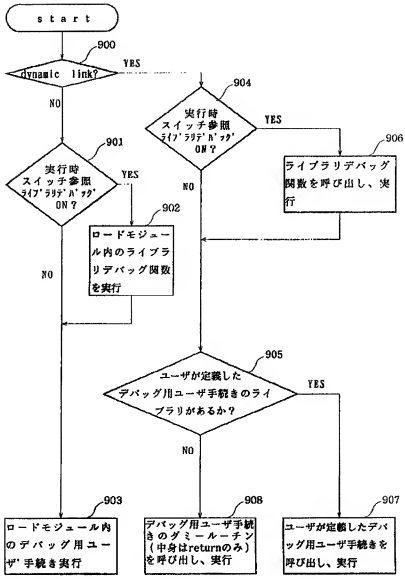
【図８】

実施例のコンパイル処理を説明するフローチャート



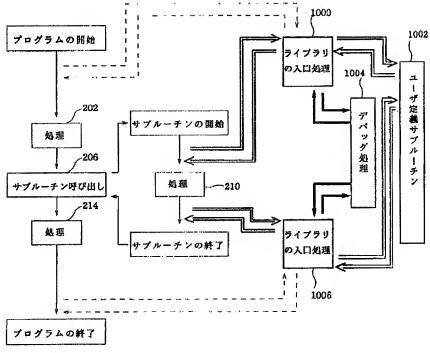
【図９】

実施例のプログラム実行処理を説明するフローチャート



【図１０】

実施例の作用説明図



実線（一）部分は、プログラムの実行順序を表している。
点線（‐‐）部分は、プログラムの開始時及び終了時に、ライブラリの入口処理及び出口処理が呼び出されることを表している。プログラム終了時又は終了時の情報が不要であれば、この部分の処理は必要ない。
太線（━）部分は、ライブラリの入口処理及び出口処理からライブラリのデバッグ処理が呼び出されることを表している。このデバッグ処理を呼び出すか否かは、実行時のオプションで切り分けることができる。
二重線（═）部分は、プログラムの翻訳時にユーザ定義サブルーチンの呼び出し処理を入れた場合の経路を表している。